curso de jquery a javascript

# La historia de jquery

jQuery es una librería de JavaScript que hizo su lanzamiento en el año 2006 con el fin de resolver diferentes problemáticas:

• Una misma forma de acceder al DOM.  
• Poder interactuar con datos de un servidor.  
• Crear animaciones.

El problema con jQuery surgió cuando se empezó a exagerar su uso y darle menor importancia a aprender JavaScript. Esto genero malos hábitos de aprendizaje y hasta en algunos casos no diferenciar jQuery de JavaScript.

A la par de que algunos se quedaban en jQuery, estaba ocurriendo la revolución de JavaScript trayendo consigo librerías que resolvían problemas específicos.

Aun con estas nuevas librerías, seguía sin resolverse el problema de hacer que uno aprenda más JavaScript que librerías o frameworks pues uno igual puede abusar de Vue, React o Angular.

Por eso en este curso aprenderás a NO depender de ninguna librería, las ventajas de esto son:

• Reutilizar conocimientos.  
• Poder implementar soluciones sin dependencias.  
• Estar más capacitado para las grandes empresas.

# Variables y Funciones

En la primera parte de este curso vamos a buscar traer datos de un servicio externo, para ello vamos a utilizar la mezcla de varias combinaciones: Promesas, ajax/fecth y funciones asíncronas. Antes de implementar una Promesa debes saber dos cosas necesarias: Variables y Funciones.

Dentro de JavaScript tenemos tres formas de declarar una variable las cuales son: var, const y let.

* var era la forma en que se declaraban las variables hasta ECMAScript 5.
* const y let es la forma en que se declaran las variables a partir de ECMAScript 6, const sirve para declarar variables que nunca van a ser modificadas y en -  
  cambio let son variables que pueden ser modificadas.

Las funciones son piezas de código que puedes reutilizar y se declaran con la palabra function.

# Promesas

Las promesas sirven para manejar nuestro código asíncrono.

“Una Promesa es un objeto que representa la terminación o el fracaso eventual de una operación asíncrona”, o dicho de forma más cotidiana, se va a mandar una función para ver si falla o se ejecuta con éxito.

Al crear una Promesa debemos pasarle por argumento la función que vamos a ejecutar de forma asíncrona, dicha función va a recibir dos parámetros para evaluar si se ejecutó bien la función o si fallo.

Nuestra promesa va a tener dos métodos para saber si todo salió bien o fallo. El método then se encarga cuando la promesa se cumplió exitosamente, mientras que el método catch se encarga cuando la promesa falla.

Dentro de JavaScript tenemos dos funciones para ejecutar una función después de algún tiempo, estas funciones son:

• setInterval: ejecutara una función cada x tiempo.  
• setTimeout: ejecutara una función después de x tiempo.

Si queremos resolver varias promesas a la misma vez, Promise cuenta con un método llamado all que recibe un array de promesas como parámetro. Este método se termina cuando todas las promesas del array se terminan de ejecutar. Si una de las promesas falla entonces el método all saltara un error mandándote al método catch.

Promise también cuenta con el método race que te regresa los resultados de la promesa que termine primero.

# Tutorial de Ajax en jQuery y Javascript

Una característica muy solicitada en cualquier sitio dinámico es solicitar datos a un servidor, denominado API. Para esto normalmente se utiliza Ajax.

Ajax recibe dos parámetros los cuales son la url de la API y un objeto donde pondrás la configuración que se usara para realizar la petición. En la configuración se añaden dos funciones para manejar cuando la petición se realizó correctamente y cuando falla.

JavaScript internamente cuenta con una función llamada fetch que también realiza peticiones a una API. Al igual que Ajax necesita dos parámetros, una url y una configuración, pero si solo le mandas la url fetch usará una configuración por defecto donde el método HTTP será GET.  
fetch te regresa una promesa, esa promesa al resolverse te da los datos de respuesta y tiene un método llamado json que te regresa otra promesa con los datos en formato JSON.

Las promesas resuelven el problema del Callback Hell haciendo que una promesa pueda devolver otra promesa y en lugar de ser anidadas como los callback, estas promesas son encadenadas.

# Funciones asíncronas

Una función asíncrona va a ser como una función normal, pero poniendo código asíncrono de forma que sea más fácil de leer de forma síncrona.

Para declarar una función asíncrona se usa la palabra reservada async, luego de eso declaras tu función de forma normal. Dentro de una función asíncrona cuentas con otra palabra reservada llamada await, lo que hará esta palabra es indicar que se debe esperar a que termine de ejecutarse ese fragmento de código antes de continuar.

Vamos a realizar peticiones con fetch a la API de yts para pedirle películas según su categoría y mostrarlas dentro de PlatziVideo. Sin el uso de funciones asíncronas para cada fetch tendríamos que usar los métodos then y catch, en cambio gracias a async/await solo debemos escribir la palabra await antes de cada promesa.

# Selectores

Un selector nos sirve para poder manipular un objeto del DOM, puedes buscar dicho objeto ya sea por su id, clase, atributo, etc.

Para PlatziVideo necesitamos un selector de un contenedor para ponerle dentro la lista de películas.

En jQuery hacemos un selector de la siguiente forma:

const $home = $(‘ .home ’);

Por convención una variable que este represente un objeto del DOM lleva el signo $, esto es para tener claro que estamos manipulando un objeto del DOM y no algún tipo de información o dato.

Dentro de JavaScript existen distintas funciones para hacer selectores, entre ellas se encuentra:

• getElementById: recibe como parámetro el id del objeto del DOM que estás buscando. Te regresa un solo objeto.

• getElementsByTagName: recibe como parámetro el tag que estas buscando y te regresa una colección html de los elementos que tengan ese tag.

• getElementsByClassName: recibe como parámetro la clase y te regresa una colección html de los elementos que tengan esa clase.

• querySelector: va a buscar el primer elemento que coincida con el selector que le pases como parámetro.

• querySelectorAll: va a buscar todos los elementos que coincidan con el selector que le pases como parámetro.

# Creación de templates

Vamos a crear una plantilla con nuestro elemento base, dicha plantilla será recibirá valores dinámicos.

Dentro de jQuery, la creación de un template seria con un texto base y si nuestro texto cuenta con distintas líneas más aparte tuviera valores dinámicos se vería de la siguiente forma:

‘<div class=”container”>’ +

‘<p id=’+ id +’>Hola Mundo<p>’ +

‘<div>’

Desde ECMAScript 6 contamos con una nueva característica llamada template literals que se representan con las comillas invertidas ``, el ejemplo anterior pasaría a verse de esta forma:

`<div class=”container”>

<p id=${id}>Hola Mundo<p>

<div>`

# Creación de DOM

La plantilla que creamos la clase anterior de momento es puro texto, no es un elemento HTML que podamos poner dentro del navegador pues si los imprimimos en el navegador lo único que veremos es texto.

Vamos a insertar la plantilla dentro de nuestro container, para ello recuerda que JavaScript se lee de arriba hacia abajo entonces debemos declarar la variable del container antes de llamar a algún método de este.

Para convertir nuestra plantilla de texto a un Document Object Model necesitamos crear dentro de memoria un documento HTML, esto es posible gracias al método document.implementation.createHTMLDocument. A este documento HTML le vamos a añadir al body, mediante innerHTML, nuestra plantilla de texto. Una vez añadida le pedimos al body el primer elemento hijo que tenga y este lo añadimos a nuestro container.

Este flujo es la magia que hay detrás de varias librerías y frameworks que nos ayudan a crear interfaces.

# Eventos

Toda aplicación web necesita lidiar con interacciones del usuario, desde un click hasta arrastrar algún elemento, estas interacciones son escuchadas por el navegador mediante algo llamado eventos. Existen muchos tipos de eventos, el más común es el evento de click.

En esta clase vamos a trabajar con el evento click y submit.

Para que un elemento HTML pueda escuchar algún evento debemos usar el método addEventListener. Este método recibe dos parámetros, el nombre del evento que va a escuchar y la función que se va a ejecutar al momento de que se accione el evento.

La página se recarga al momento de ejecutarse el evento submit, para evitar esto debemos quitarle la acción por defecto que viene en submit usando el método event.preventDefault().

Son una forma de notificar a la aplicación cuando algo interesante ha sucedido.

**jQuery**

$("div").on("click", **function**(event) {

})

**JavaScript**

**const** $element = document.getElementById("element");

$element.addEventListener("click", **function**(event) {

})

**Nota**: cuando se activa el evento submit, el browser de refresca por defecto. Para evitar esto se usa event.preventDefault().

Para ver la lista de eventos:  
<http://developer.mozilla.org/en-US/docs/Web/Events>

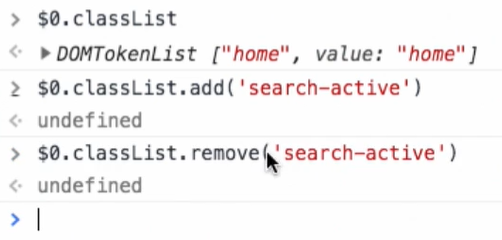
# Clases y estilos CSS

En esta clase vamos a aprender a manipular las clases de CSS y estilos de nuestros elementos mediante JavaScript.

Dentro de cada elemento tenemos un método llamado classList, con este podemos ver las clases que tiene nuestro elemento y además llamar a otros métodos para añadir, borrar o hacer toggle a alguna clase.

De igual forma podemos acceder a todas las propiedades de CSS algún elemento mediante element.style.

Inspecto de google chrome



# Creación de elementos y asignación de atributos

Vamos a crear un elemento HTML sin usar un template string. Para crear el elemento desde cero vamos a usar el método document.createElement, este recibe como parámetro la etiqueta html del elemento que se quiere crear, no funciona mandándole el template string.

Para añadirle un atributo al elemento que acabamos de crear haremos uso del método setAttribute. Este recibe dos parámetros, uno indicando el nombre del atributo que vamos a añadir y el segundo parámetro indicando el valor de dicho atributo. Vamos a crear una función para poder añadir múltiples atributos a un solo elemento.

## Creación de elementos

**const** $loader = document.createElement("img");

## Asignación de Atributos

**jQuery**

$("#element").attr({

src: "",

height: ""

})

**JavaScript**

**const** $element = document.getElementById("element");

//setear el atributo en un elemento DOM

$element.setAttribute("src", "img/foto.png");

//obtener un atruto de un elemento DOM

**const** src = $element.setAttribute("src")

También se puede crear una función para asignar múltiples atributos a un elemento DOM.

**functionsetAttributes**($element, attributes) {

**for**(**const** attribute **in** attributes) {

$element.setAttribute(attribute, attributes[attribute]);

}

}

# Formularios

FormData() es una interfaz que te permite obtener los valores de un formulario.

//FormData va a abstraerr todos los valores de los elementos del formulario que cuenten con un atributo 'name' asignado y los va a setear en un objeto de tipo FormData.

**const** data = **new** FormData($form);

//retorna el valor del elemento con el atributo name="nombre"

data.get("nombre");

//setea el valor avengers en la key pelicula

data.set("pelicula", "avengers");

# Desestructuración de objetos

Destructuring assignment permite entrar a un objeto o lista y poder sacar un dato para asignarlo a otra variable.

//el fetch devuelve una promesa con la siguiente estructura: promesa.data.movies

//con el destructuring assignmen estamos creando una variable que se llama pelis y solo contiene la información de movies.

**const** {

data: {

movies: pelis

}

} = **await** fetch(`api\_url`);

//Lo anterior sería igual a esto:

**const** response = **await** fetch(`api\_url`);

**const** pelis = response.data.movies;

# Dataset

Dataset permite acceder a un objeto con todos los atributos data de un elemento DOM.

<divid="element"data-id="10"data-category="action">

</div>

**const** $element = document.getElementById("element");

//guarda el valor de data-id

**const** id = $element.dataset.id;

//guarda el valor de data-category

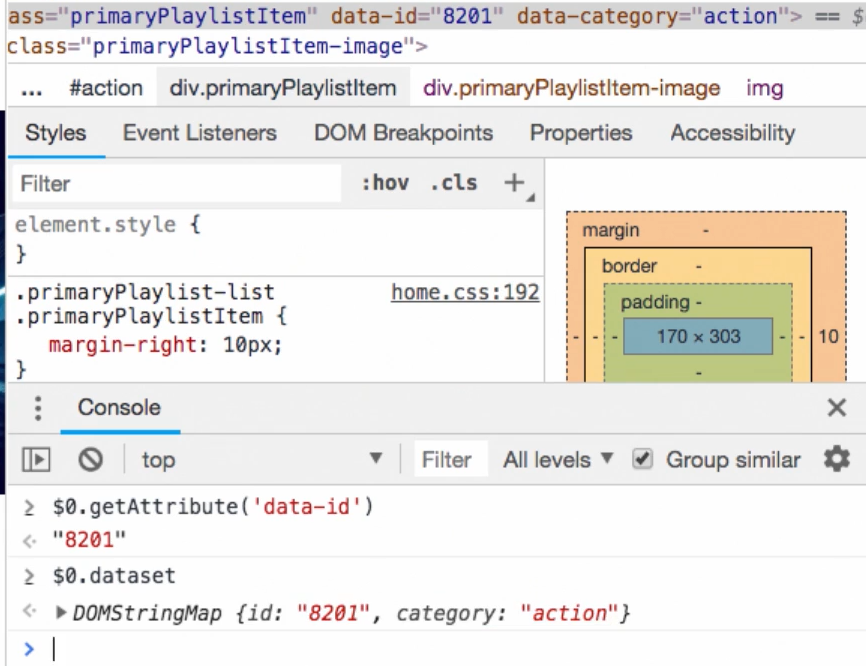
**const** category = $element.dataset.category;

## Transformar tipos de datos

Cambiar un string a entero

//parseInt("número", base)

**let** n = parseInt("500", 10)



# Encontrando elementos en lista (find)

find() devuelve el primer elemento de un array que cumpla con el criterio de búsqueda. Si no se encuentra ningún elemento devuelve undefined.

**functionfind**(list, id) {

**return** list.find(movie => movie.id === id)

}

Información referencial:  
<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/find>

## switch

**switch** (category) {

**case**"action": {

//código de action

}

**case**"drama": {

//código de drama

}

**default**: {

//código por defecto

}

}

# Manejo de errores

El manejo de errores se hace con un bloque try/catch. Se intenta ejecutar un bloque de instrucciones (try) y se especifica una respuesta en caso suceda un error (catch).

**try** {

//codigo a evaluar

}

**catch**(error) {

//código por si sucede un error

alert(error.message);

}

* Se puede crear un error customizado con Error().
* Se puede lanzar un error con throw.

**thrownew**Error('No se encontró ningún resultado');

# Guardar datos

* localStorage permite almacenar datos sin tiempo de expiración
* sessionStorage permite almacenar datos. Estos datos se van a borrar cuando se termine la sessión del navegador

En local storage solo se puede guardar texto plano. No se pueden guardar objetos.

//eliminar los datos

window.localStorage.clear();

//setear un valor

window.localStorage.setItem("nombre", "Toshi");

//setear un objeto

//primero se tiene que convertir el objeto en un string

window.localStorage.setItem("objeto", JSON.stringify({"peli": "wonder woman"});

//obtener el valor de un key

window.localStorage.getItem("nombre");

//obtener el valor de un texto objeto y convertirlo a objeto

JSON.parse(window.localStorage.getItem("objeto"));

# Obteniendo los datos almacenados

Revisar si los datos se encuentran en cache.

**asyncfunctioncacheExist**(category) {

**const** listName = `${category}List`;

**const** cacheList = window.localStorage.getItem(listName);

**if** (cacheList)

**return**JSON.parse(cacheList);

**const** { data: { movies: data } } = **await** getData(`${BASE\_API}list\_movies.json?genre=${category}`);

window.localStorage.setItem(listName, JSON.stringify(data));

**return** data;

}

Si se desea volver a traer los datos se puede hacer lo siguiente:

* Poner un botón que traiga los datos
* Hacer un setTimeout que borre el localStorage.